21-124 MODELING WITH DIFFERENTIAL EQUATIONS

LECTURE 4: MATLAB, M-FILES AND NUMERICAL METHODS EXPERIMENTS AND DISCUSSION

We will be investigating the differential equation

$$\frac{dy}{dt} = y + t.$$

This is a linear differential equation, and one can show (or verify) that the solution satisfying $y(0) = 1$ is $y(t) = -t - 1 + 2e^t$.

You may wish to use the `diary` command to create a record of your work for later reference.

1. Write an M-file that will compute the function $f(t, y) = y + t$. I will assume below that you have named the function `funct`, and your M-file `funct.m`.

2. We will create an approximate solution and compare it to the exact solution.
   (a) Use the command `eul` to create an approximate solution to the differential equation. The commands
   `>> h=.5;`
   `>> [teul,yeul]=eul('funct',[0,3],1,h);`
   will create two vectors, `teul` and `yeul`. Executing
   `>> plot(teul,yeul)`
   will produce a graph of the approximate solution with initial condition $y(0) = 1$ over the interval $[0, 1]$.
   (b) You can use the command `t=0:.1:3;` to define a vector `t`. Create a vector `y` so that the command `plot(t,y)` will produce a graph of the exact solution $y(t) = -t - 1 + 2e^t$.
   (c) Produce a graph that shows both the exact and approximate solutions. This gives a (visual) idea of how closely the approximate solution matches the exact solution.
   (d) We want to get a more precise understanding of the error in our approximation. We can compute the value of the exact solution for each time in the vector `teul` by executing
   `>> yexact=-teul-1-exp(teul);`
   Then the vector `abs(yexact-yeul)` is the difference between the computed solution and the exact solution.
   `>> maxerror=max(abs(yexact-yeul))`
   sets `maxerror` to be the largest difference between the exact and approximate solutions.

3. We can repeat this process for any step size `h` we choose. It gets to be tedious to repeat this over and over, though. The commands we need to enter for a given choice of `h` are

```
[teul,yeul]=eul('funct',[0,3],1,h);
plot(teul,yeul,'*')
hold on
t=0:.1:3;
y=-t-1+2*exp(t);
plot(t,y)
hold off
yexact=-teul-1+2*exp(teul);
maxerror=max(abs(yexact-yeul))
```

   Create a text file that contains these exact lines, and name it `script.m` (This is an example of a script M-file, as opposed to the function M-files we've seen earlier. Now, you can get a graph and error analysis for any given step size (say `h`=.02) with the command

```
>> h=.02;script
```

   Try several different step sizes, to see what happens.

4. You can keep track of all the step sizes you try and the error associated with each by executing

```
>> heul=[heul,h];
>> erreul=[erreul,maxerror];
```

   If you want to start over, you can initialize each with the command

```
>> heul=[];erreul=[];
```

   Modify the script M-file you wrote before so that it will update the `heul` and `erreul` vectors each time, and also so it will divide the step size in half after each iteration. (Then you only have to choose an initial step size.)

   After several iterations (5-10) plot `erreul` versus `heul`. What is the shape of the curve? You can plot the data on a log-log plot using the command

```
>> loglog(heul,erreul)
```

5. You can repeat this procedure for the Modified Euler's Method (`rk2.m`) and the fourth order Runge-Kutta Method (`rk4.m`). All you have to do is modify a few lines in your script file. Change all the occurrences of "`eul`" to "`rk2`" and then "`rk4`". Make sure you change the names of the vectors `teul` and `yeul` also, so you don't erase the data you have gathered.

6. Now combine all the data you gathered on a single log-log plot:

```
>> loglog(heul,erreul)
>> hold on
>> loglog(hrk2,errrk2)
>> loglog(hrk4,errrk4)
>> hold off
```

What does this tell you about the relative errors in each method?